Start trusting Your BIOS - SRTM with vboot, TPM and permanent flash protection

Piotr Król[1], Michał Żygowski[2], and Łukasz Wcisło[3]

[1] 3mdeb Embedded Systems Consulting, Gdansk, Poland
piotr.krol@3mdeb.com
[2] 3mdeb Embedded Systems Consulting, Gdansk, Poland
michal.zygowski@3mdeb.com
[3] 3mdeb Embedded Systems Consulting, Gdansk, Poland
lukasz.wcislo@3mdeb.com

Abstract

In this paper, we are going to introduce Static Root of Trust Measurement with Verified Boot using different mechanisms of SPI flash protection. We shall prove VBoot great support for coreboot, TPM usage, and cryptographic operations, and its ability to perform measured and verified boot. We will explain why the Root Key and Recovery Key are the most important components in the VBoot and should be well protected. As a result, we will show a mechanism for automatic decryption of a disk with the assistance of TPM and policies tied to the firmware measurements stored in the TPM.

1 Introduction

SRTM - Static Root of Trust for Measurement - is the technology that make the entire trust begin with the static, immutable piece of code, which is called Core Root of Trust for Measurement (CRTM) [1], and which would measure the BIOS itself with Trusted Building Block (TBB) that remains unchanged during the lifetime of the platform.

Currently supported firmware measurement and verification methods in coreboot are developed by Google reference verified boot implementation called VBoot with addition of TPM usage to measure firmware components.

2 Static Root of Trust for Measurement goals

The purpose of such measurement is verification of device firmware components. If for some reason there were some unauthorized changes in our firmware (i.e. in a result of a malicious attack) the device won't boot and we will see that the component hashes in the TPM log changed (for the sake of not letting anyone execute some dangerous code). There are countless ways to attack someone's device by infecting its firmware, but that isn't the scope of this paper. Let's just say it may cause a severe problem.

The idea behind the whole concept is: measure, verify and then execute. So basically all the components used in the boot process should be measured and verified, in particular AGESA, PSP firmware, coreboot stages, ACPI and SMBIOS tables.

3 Verified Boot

The VBoot project was created for the purpose of securing a Chromebook firmware. The main goals [2] were:

1. Detect non-volatile memory changes from the expected state (RW firmware)
2. Detect file system changes relevant to system boot (kernel, init, modules, filesystem metadata, policies)
3. Support functionality upgrades, easy firmware update, and recovery

3.1 Flash image layout

coreboot ROM image is not just a monolithic block, but it contains named objects. This is CBFS, the coreboot read-only File System used to place components inside the image during build process as well as identify and locate components during boot process. Google has developed a new solution to represent the layout of coreboot flash images. Basically, it is a structure that defines regions in flash memory (their base addresses and sizes) that lies outside the CBFS layout in the specified region. This binary format describing partitions in a flash chip is called flashmap.

What is important here is that flashmap can support multiple CBFSes inside one firmware image. It gives a great opportunity for various firmware updates. As a typical VBoot implementation, the image has a read-only partition for recovery purposes and the read-write firmware A and B partitions. The reasoning of having two read-write partitions is that when one updates partition A and the platform is not booting, one may switch to partition B containing older working code. In such a way one may prevent bricking the device with flashing incorrect firmware. Thanks to the flashmap firmware updates are easy and can target only specific components, see Figure 1 for more details.

4 Core Root of Trust for Measurement

The CRTM in the VBoot implementation lies in the read-only recovery firmware components self-measurement. To these measured components we include:

1. the bootblock (component placed at reset vector),
2. Google Binary Block (GBB) with public part of Root Key used to verify other keys
3. VBoot verstage (if separated from romstage)
4. romstage (if VBoot starts in romstage)

The Root Key stored in GBB is also composed in a key block along with firmware signing key in the read-write firmware. The firmware signing key was used during build time to sign the firmware partition. Thus the Root Key is one of the most important part of the image.

In the Chromebooks firmware, this region with recovery firmware is protected by hardware SPI flash protection which prevents any writes or erases to the region. The protection is enabled with few block protection bits in the SPI flash status register, status register write protection bit and a grounded write protection pin (a screw or a jumper on the motherboard). In such a case, the recovery region is immutable (without physical access to the hardware) and thus can store important data.

In our solution, we also implement CRTM with Vboot self-measurements and the SRTM in the immutable SPI flash, but in contrary to Google implementation the whole read-only recovery partition lays under permanently locked SPI flash area. That means, that even having physical access to the device it cannot be modified without desoldering SPI flash chip. That makes our SRTM reasonably safe.

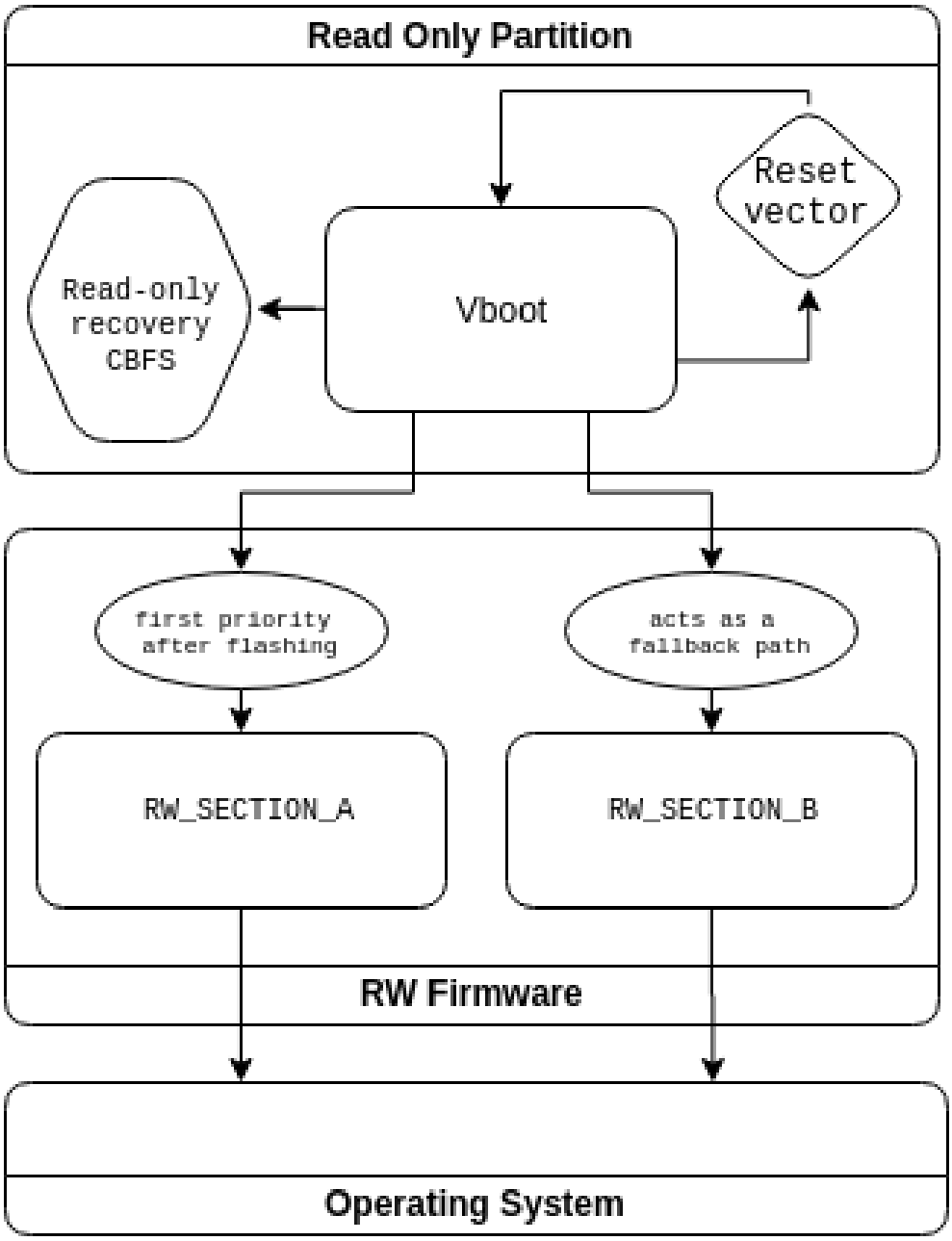

Figure 1: Verified Boot diagram.

5 Advantages of VBoot

First of all VBoot contains all necessary cryptographic functions implemented. One does not need to worry about implementing common hashing algorithms.

Secondly, VBoot enforces each firmware partition to be digitally signed by firmware signing

key residing in the VBLOCKs in the RW firmware. The VBLOCKs also contain hashes of the firmware partitions, they belong to. Vboot verifies those and prevent from executing the firmware if verification of the VBLOCK fails.

VBoot requires certain components to be present, like GBB, FMAP (flashmap descriptor region), VBLOCKs, read-write firmware slots. With the flashmap layout one can obtain multiple firmwares inside one binary on SPI flash. What is more flashmap makes it easy to update single regions/firmwares unlike typical proprietary firmwares.

VBoot contains firmware version anti-rollback protection, which prevents from malicious firmware downgrade by an attacker. The oldest allowed version is stored in TPM non-volatile memory managed by VBoot. If the version of firmware is older than the version stored in TPM non-volatile storage, it will not be executed.

VBoot has many recovery reasons and methods implemented. If there is any error with firmware verification or secure TPM non-volatile memory is corrupted, VBoot logic invokes a recovery and resets the platform. The purpose of this mechanism is to let the user inform about an error or attack and possibly recover the firmware to trusted state.

Vboot is open-source and easily integrable with coreboot. Ensures only correctly signed firmware executes bringing platform to operational state, allows firmware behavior customization with variety of target platform user-space tools.

6 Core Root of Trust protection

Typically hardware write protection of SPI flash is sufficient to prevent most of the attacks targeting the firmware. However, protecting the SPI flash just with WP pin may not be enough. Anybody with a screwdriver and 5 minutes of time may bypass the write protection and rewrite whole firmware and destroy the Core Root of Trust[4]. Like other solutions, it is worth considering permanent SPI flash lock for read-only region. In such a way, the only method would be to solder a new SPI flash with attacker's own firmware or modifiable copy of previously dumped original firmware. Although the permanent lock takes away the freedom of firmware modification, it was especially designed to fulfil the customers requirements for secure firmware.

There are basically 3 mechanisms found on SPI flashes to lock writes/erases to the SPI flash regions:

1. Hardware status register protection with block protection
2. Permanent hardware status register protection with block protection
3. Volatile and Non-volatile Block Protection Register Lock

Below we are going to briefly describe them, but it is a really broad subject and out of the scope of this paper.

6.1 Hardware status register protection with block protection

Each common SPI flash has block protection bits (typically 4) to lock a certain number of blocks in the flash memory. The blocks can be chosen counting from the beginning of the memory or from the end of memory. The locked space is always contiguous and starts either from the end or from the start of memory (not in the middle of flash). The block protection bits are located in the status register. They can be changed as long as the status register is not protected. To

protect status register, manufacturers have designed status register write protect bit (SRP0) and a status register write protect pin (WP). This protection mode is secure only when nobody has access to the WP pin (or it is permanently tied to the ground on motherboard). This is the most common write protection feature for SPI flashes present on Macronix MX25L6406E [10].

Protection level	SRP state	WP state
No protection	0	X
Software protection	1	1
Hardware protectio	1	0

X - don't care

Table 1: Hardware status register protection with block protection

For SRTM implementation it can be dangerous since one can remove the screw or jumper and then manipulate the CRTM to give a false feeling of trusted state of the platform. Attacker could even inject own public Root Key to execute own signed firmware.

6.2 Permanent hardware status register protection with block protection

Certain chips have a secondary status register which contains second Status Register Protect bit (SRP1). This bit allows setting two additional protection modes:
Power Supply Lock Down - Block protection is immutable until the power cut off. Status register cannot be altered in any way until next power down and power up cycle of the chip after SRP1 (Status Register Protect bit) was written to 1.
One Time Program - once SRP1 and SRP0 bits are set to 1 there is no other way to change the status register. Previously set block protection bits are immutable and protected blocks are permanently locked. There is no known way back from this state. The only possible method to change the firmware in the protected region would be chip desoldering and exchange.

Both features are available on special order [9].

Protection level	SRP0 state	SRP1 state	WP state
No protection	0	0	X
Software protection	1	0	X
Hardware protection	1	0	0
Power Supply Lock Down	0	1	X
One Time Program	1	1	X

X - don't care

Table 2: Permanent hardware status register protection with block protection

Since this is the simplest and less complicated method, we decided to choose this one. It provides the best read-only partition security by permanently locking this section. The only attack possible here is chip exchange, which is a sophisticated attack and requires time and special tools. Only desperate and well prepared attacker can afford that.

6.3 Volatile and Non-volatile Block Protection Register Lock

There are chips like SST26VF064B from Microchip which have an additional registers for block protection and different approach of locking the SPI flash regions. The SST26VF064B invents 3 registers which control access: the status registers, control register, and Block Protection Register (BPR) containing block protection bits as in previous options.

The theory of operation is different than in previous mechanism. The status register contains Write Protection Lock Down bit (WPLD) which indicates the availability of access to the BPR. Whenever WPLD bit is set, one cannot write to BPR using the special command for locking the SPI flash not present in previous options. To set the WPLD bit one has to issue another special locking command after the BPR register has been set with desired protection bits. This command locks only the volatile part of block protection. The lockdown will be reset after a power cycle of the chip. There are also a non-volatile parts of the block protection bits which can be permanently set with special commands. The most interesting part of this solution is that these block protection bits can restrict reads and writes to the regions corresponding to the given bit. One may lock any region and is not restricted to choose the range either from beginning or the end of flash space. For more details refer to the SPI part datasheet [8].

This variant gives the best options to lock the flash. However, it is still vulnerable to desoldering and chip exchange attacks. The advantage of this solution is that we can protect single parts of the read-only partition, like bootblock, verstage and GBB which are the most important. It still leaves the other components updatable, for example fixed offset AGESA binary etc.

7 Disk encryption with SRTM

Given that one has a platform booted with trusted firmware. TPM contains the BIOS measurements in its Platform Configuration Registers (PCRs). They can be used to seal and unseal secrets in TPM non-volatile memory. We will not go into details in this document, but would like to describe the overall concept of TPM usage in disk encryption and decryption process.

Each disk encryption process requires setting a password that is used when decrypting the disk. This password can be saved in TPM non-volatile memory and sealed using the selected PCR values of trusted firmware. The advantage of such approach is that the password will not be unsealed from TPM non-volatile memory unless the selected PCRs matches those from sealing operation. In short, only when the trusted firmware is booted, the password can be retrieved and used to decrypt the disk.

In our solution we utilize TPM capabilities to seal the disk encryption password in the TPM non-volatile memory and the Linux drivers to automatically unseal the password in the initial ramdisk to be passed to the decryption software. In such way one may obtain a fully secured firmware/user-space stack protecting the sensitive and important user data on disks from firmware attacks.

8 Future improvements

There are many opportunities to improve the solution in various ways and a few limitations that would be great to be fixed.

For now, the PC Engines apu2 platform[5] has no C environment bootblock, which causes the VBoot being executed in romstage as its integral part. Having the C bootblock would open a way to a separate verstage and moving the romstage to the updatable read-write firmware

sections. It is also crucial for platform initialization, since many calls to AGESA are done in romstage actually.

Secondly AGESA must be placed at certain fixed offset which unfortunately lies close to bootblock, thus it has to stay in locked read-only partition. If one would like to update AGESA binary, it is currently impossible. From what we know AGESA does not have a tool to rebase the binary like Intel FSP does have.

TPM 1.2 are quite widely supported in Linux and BSD systems. What we would like to have is the TPM 2.0 supported in our solution both in Linux and FreeBSD. The main advantage of TPM 2.0 is the availability of SHA256 algorithm in contrary to TPM1.2 which can utilize SHA1 only. SHA1 algorithm is no longer considered secure since the collisions of full SHA1 has been proven [3].

Another interesting feature is usage of OPAL disk. This is a secure, self-encrypting disk (SED) which is compliant with the OPAL disk specification designed by Trusted Computing Group (TCG). The gain from this approach is to let hardware do the decryption/encryption to not slow the operating system software or provide additional safety barrier to prevent more attacks. Typically the disk can be unlocked by unsealing the key from TPM (in case of normal disk) or provide a SED unlock password for OPAL disk. But taking the both options together may create a higher protection level. However, since TCG OPAL reference specification is open, anybody can make their own OPAL disks. Many implementations have their own specific bugs, which already have been proven [7].

What is more, current key management in VBoot is limited to generating a fresh RSA keypairs and packing them into special VBoot key packs. All is done by an utility called vbutil, which is also responsible for signing the image. Since vbutil integrates many functions from OpenSSL, we think it will be possible to have the PKCS11 standard used to sign the firmware partitions with Firmware Signing Key stored on a Cryptographic Token like NitroKey [6] or similar. In such case the keys are much less vulnerable to leaks and unauthorized usage by an attacker who could steal the keys to produce signed, trusted firmware for Your platform.

9 Summary

VBoot is an easy integrable firmware verification library. It has great support for coreboot, TPM usage, and cryptographic operations. It is able to perform measured and verified boot based on the hardware protected Static Root of Trust for Measurement in the ready only SPI flash.

The Root Key, bootblock and verstage are the most important components in the VBoot and should be well protected.

SRTM opens a path for secure firmware measurement and verification which can lead to trusted decisions, like unsealing the disk encryption password from TPM.

In our consideration, there is nothing else left, than to recommend any user that put emphasis on the protection of firmware from malicious attacks, using VBoot and its utilities.

References

[1] David Cooper, William Polk, Andrew Regenscheid, Murugiah Souppaya, *Recommendations of the National Institute of Standards and Technology,* Special Publication 800-147 , National Institute of Standards and Technology, April 2011

[2] https://www.chromium.org/chromium-os/chromiumos-design-docs/verified-boot *The Chromium OS Design Documents, Verified Boot* (date of access: 24.06.2019)

[3] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, Yarik Markov, *The first collision for full SHA-1* CWI Amsterdam, Google Research, February 2017

[4] https://youtu.be/loBX_vEXxVA

[5] https://www.pcengines.ch

[6] https://www.nitrokey.com/

[7] Carlo Meijer, Bernard van Gastel, *Self-encrypting deception: weaknesses in the encryption of solid state drives (SSDs)* Radboud University, the Netherlands, November 2018

[8] https://ww1.microchip.com/downloads/en/DeviceDoc/20005119G.pdf

[9] https://www.winbond.com/resource-files/w25q64fv

[10] https://www.macronix.com/Lists/Datasheet/Attachments/7370/MX25L6406E